AD 673333

COMPUTER SOFTWARE: THE EVOLUTION WITHIN THE REVOLUTION

George W. Armerding

July 1968

P-3894

15

COMPUTER SOFTWARE: THE EVOLUTION WITHIN THE REVOLUTION

George W. Armerding*

The RAND Corporation, Santa Monica, California

A corporate treasurer signs a check for $30,000, in payment for a five year lease of a proprietary computer program. The United States Patent Office issues a patent protecting the inventor of a computer sorting technique, represented in the patent by a program flow chart. A computer manufacturer announces availability of his COBOL compiler at extra cost -- customers who want it, must pay for it.

The above incidents are signs of the evolution taking place within the realm of computer software. I call it an evolution, to differentiate it from the revolution continually seen in computer hardware: speeds have gone up, prices of hardware have gone down, and the price of the execution of a single instruction is diminishing so rapidly engineers now talk about "throwaway computers" which will be cheaper to replace than to repair.

But within this continuing hardware revolution we see slower evolutionary changes taking place within the software

---

*Any views expressed in this Paper are those of the author. They should not be interpreted as reflecting the policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

realm. These changes, while not as exciting and visible as those in hardware, are still significant -- perhaps more significant.

The Hidden Cost Begins to Show Itself

Twist the arm of the computer manufacturer's cost accountant and he will admit it; the cost of developing and maintaining the software for his current family of computers has approached, perhaps exceeded, the cost of developing, manufacturing, and maintaining the hardware on which that software runs.

As we all know, the manufacturer gives away his computer software. It's "free." Yes, the marginal cost of *getting the software from* the manufacturer -- once you have bought his hardware -- is indeed zero. But no matter how you evaluate it, half of what you pay that manufacturer is for the software, not for the neat, colorful boxes of electronic components. The customers and manufacturers are becoming aware of this hidden cost and are beginning to take action to control it.

At least one computer manufacturer now offers his COBOL compiler as an "extra." If you don't want it, don't pay for it.

Within the ranks of computer users, we are beginning to hear rumblings that manufacturers should price <u>all</u> of their software separately. Then the customer can go down the price list and pick out -- and pay for -- only the items he needs.

This mode of operation could have some interesting economic effects on both the manufacturer and the customer.

The Programmer is Not a Programmer

Consider the evolution which is taking place among people who write computer programs. In the early days of computers, it took skill, perseverance, and a certain amount of alchemy to be a programmer. Computers were instructed by an elite group often drawn from strange and diverse backgrounds.

Who programs now? Everybody. It seems to be a point of honor to be able to advertise that "an hour's worth of simple instruction can make every member of your engineering or accounting staff a qualified user of our online timesharing system." This evolution away from "Closed Shop" programming by the professional elite, into "Open Shop" programming by everyone has led to some interesting speculation. Some suggest that professional programmers will disappear. Others suggest that they won't disappear, but will become very specialized, suited only to perform rare and exotic programming such as writing compilers or complicated service subroutines.

On the other side of the argument, the purists assert that we will always have professional programmers. The programming discipline will never sink to such a low common denominator that anyone will be able to do it. Yes, there will be on-line dilettantes -- the one-hour wonders -- but the hard core of "real" programming will always be done by the professionals.

It appears that these arguments are at once both right and wrong. Yes, much programming will be done by Open Shop users at typewriter consoles. Yes, specialized programming will continue to be done by the programming professionals. And it will probably always be true that there will be more

programs to be written, at all skill levels, than there will be people at those skill levels capable of writing them.

But the interesting speculation is not whether programmers as a group will disappear, but whether we will experience a growth of a phenomenon which we now see; many people who would not define themselves as programmers nevertheless spend a majority of their working hours doing the sorts of things which professional programmers would categorize as programming.

Consider the open shop programmer who becomes intrigued by his online terminal, to the point where he spends a large portion of each working day trying to make his program work properly, or adding a new feature, or trying its behavior with slightly different data. He is a programmer. He soon tires of playing with the machine, and gradually turns his computing chores over to a technical aide. The aide, who formerly worked a desk calculator and plotted points on graph paper, now spends a major portion of his day seated at the master's console, tracking down bugs and making changes according to the boss's wishes. The aide is a programmer.

Consider the business executive of the future who has a CRT display installed in his office. It shows him up-to-the minute sales data, allows projection of wage and salary costs five years into the future; and gives him a quick look at the personnel file of a young man being considered for a promotion.

When the CRT was first installed, the executive used it himself. In fact, he developed a modicum of skill at making it honor some reasonably complicated requests.

But this will soon become boring. He will hire a fresh young MBA with a couple of years of computer courses on his college record. That young man will then be charged to instruct the console to perform special tasks at the executive's bidding. The young MBA is a programmer.

People who call themselves programmers may begin to disappear. But they will most certainly reappear in other places, referring to themselves by other appelations. And the combined population of "real" and "hidden" programmers will certainly continue to grow rapidly.

Be Inefficient and Like It

Not too long ago in the history of computers, the watchword of the programmer seemed to be "efficiency." A good programmer was one who could squeeze the last few machine cycles out of the inner loop of the program. Only incompetents would use such things as trace programs. Interpretive programs were used only with great care, because of their "inefficient' use of the computer.

Now we are in the hardware revolution, and the software world is experiencing an enlightened attitude toward apparent inefficient machine usage. As an example, consider time-sharing. Few eyebrows are raised when it is reported that a general purpose timesharing system uses 50% of the available computer time performing its various overhead operations.

Or interpretive programs. We find that even our installed compiler -- one of the most heavily used programs on our computer -- runs interpretively, making our computer behave like something it isn't. Inefficient? No. In fact, the compiler is well worth the time which it takes.

Operating systems, loaders, editing programs, and innumerable other overhead functions quite typically take a significant share of the CPU cycles away from the "real" user -- the problem program.

In spite of overhead operations which make the old programmer cringe, today's computer management, as well as the new generation of programmers, find nothing wrong with our mode of operation. To them, the benefits surpass the costs.

True, we would all like to squeeze more useful work out of our computer, but we now know that, everything considered, we are paying less -- much less -- per useful answer than we ever have in the past.

Increased hardware performance per dollar of hardware cost more than compensates for loss of the ever increasing portion of the machine's power absorbed by the software.

This trend can be expected to continue. As long as the hardware designers keep reducing the cost of computers and keep raising their performance, users will tolerate -- even demand -- more and more "overhead" software to make things easier for the programmer and his program.

Only when the hardware revolution diminishes can we expect a reversal of the present tendency to greater software overhead. Only when the computer designers have squeezed the last ounce of power out of their circuits and the manufacturers have reduced their production costs to their reasonable limits can we expect the software builders to begin worrying, in earnest, about software overhead rates.

That day appears to be far in the future.

Timesharing Loses and Wins

Programmers poke fun at the problem being experienced by implementors of large, general purpose timesharing systems. The "I told you so" attitude is prevalent. But while we sneer at (or commiserate with) the developers of general purpose timesharing systems, some little foxes have quietly been stealing the grapes. Small, limited purpose timesharing systems are proliferating. Innovative systems appear at the universities, fathered by clever graduate students. Profit-oriented systems are emerging weedlike; each day's mail seems to bring notice of one more on-line system which is available (for a price).

The advent of these smaller, more specialized timesharing systems has relieved much of the pressure from general purpose timesharing implementors. But even while the general purpose camp struggles to meet their promised specifications, the smaller systems are multiplying.

The inevitable results will be a merger. Some of today's limited systems will boast added features, languages, file systems, and programmer aids to the point where they will be indistinguishable from true general purpose systems. By then, the battle will have been won. We will, in fact, have general purpose timesharing systems.

Programs for Sale

"Let's band together for our mutual benefit," said the pioneer computer users, and they did. Early users of large computers realized that they had much more to gain by sharing among themselves than by operating behind closed doors. This resulted in the formation of user groups. One of the first

projects undertaken was the sharing of programs. Hundreds, even thousands, of computer programs performing all manner of calculations and manipulations were donated by their originators and distributed, free of cost, to all comers.

The software evolution is also at work here. The early spirit of "together we survive, apart we die," is gone. Now any program worth the cards it is punched on is being offered, not free, but for a price. Major software houses take full page ads in computer magazines offering large proprietary programming packages. In the want-ad section of a computing newspaper, a lone programmer asks $150 for a copy of his latest, improved, high speed sorting program.

The pioneers are gone; the competitors are here. The cooperative spirit of the old guard has been replaced by the business sense of the new generation of computer experts.

Early software entrepreneurs found the going rough. A few early programs offered for sale didn't find enough takers to cover development expenses. The market is now more favorable. Users who, at first, were revolted at the prospect of having to pay for something they traditionally got for nothing, recognized the inevitable. They are beginning to lay out real dollars for the privilege of using proprietary program packages.

Reckless competition is sure to come. Every programmer in the country is a potential software entrepreneur. All he needs is a pad of coding paper, some pencils (with erasers) and a few hours of moonlight time during evenings and weekends. Countless computer installations are happy to sell him machine time to debug and test his program.

Now that some of the first volleys are beginning to hit their marks, this part of the software evolution may well become a full scale revolution. Don't be surprised by an avalanche of proprietary computer programs on the market. It is not difficult to imagine the day when all software will be sold, with multiple suppliers of equivalent programs and true competition.

Some Competitive Implications

With the advent of manufacturers' separate pricing of software and of proprietary programs which are offered for sale, the door is open for significant economic changes within the realm of computer software.

Now, most programs offered for sale are applications oriented. As such, they seldom compete directly with software offered "free" by the manufacturer. But we can expect that in the not too distant future enterprising programmers will begin competing with manufacturers. The market could see improved versions of manufacturer software, or software which functionally replaces the "free" software, or programs which supplement the standard software or make it easier for users to approach.

Couple the above with an increased tendency for manufacturers to price hardware and software separately and you have the makings of a highly competitive situation. The manufacturers, of course, have the upper hand. They can adjust the price of software as they desire, ranging from "free" to profitable. But as high performance substitutes appear on the open market, even at a price, the pressure will fall on the manufacturers to either improve their performance or adjust their price.

When the user begins paying for a substitute product, the cycle of pressure will continue. Now the user will certainly want a separate hardware/software price schedule because he will not want the manufacturer's software, and will not want to pay for it, either directly or indirectly. Further separate software pricing will invite further competition; this cycle will continue until price stability occurs.

An example of this competitive situation may already exist -- or be close to existence -- in the realm of management information systems. These are programs which currently are being offered "free" by manufacturers and for a price by entrepreneurs. It will be interesting to follow this competition closely, using it as a data point to test the above speculation.

Don't Steal My Program

The automobile is a very mobile (if you will excuse the play on words) device. It is also expensive, in an absolute sense. These two factors make it a prime target for thieves. So government, in the form of laws, registration procedures, and policemen in patrol cars, takes costly, but valuable, steps to protect the automobile owner from the dishonest.

Most computer programs being offered for sale these days are priced higher than the cost of an automobile, and are significantly easier to steal.

Some programs have been copyrighted. But it is not clear that this offers effective protection to the author. Recently, the first computer program patent was issued; the

validity of this action remains to be tested in the courts. Even if the program patentability is upheld, the effectiveness of this protection is also suspect.

A positive means of protection will be required. As more and more computer programs are sold (or leased), more and more opportunity occurs for the dishonest to perpetrate his foul deeds. The nascent sinful nature of man predicts that he will.

If copyrights and patents are not sufficient, is some new form of protection indicated? Will there be special legislation designed to provide protection for computer programs? Because of the complexity and novelty of the computer milieu, it appears that protection will come only after substantial foul play puts pressure on government to "do something."

What About Hardware?

The hardware revolution will be affected by the software evolution in two important areas: reliability and machine organization.

By almost any measure, today's computers are extremely reliable. Compared to models produced five or ten years ago, they are superb. However, in timesharing use a high level of reliability is not enough. Consider a timesharing computer which operates perfectly for, say, fifty hours, then malfunctions for two seconds. Suppose that, during the two seconds, all communication between the online users (there may be several dozen) and the computer are lost; program pointers to data are confused; dictionaries of files are destroyed; and key portions of the operating software

have been disabled. Each user is going to be very, very unhappy.

If the computer were being used in a batch operation rather than in timesharing, only one or a few programmers would be affected. The operator would restart their programs from their beginnings and little notice would be taken of the failure.

In a timesharing situation, however, each of the several dozen users will be extremely unhappy. Each has lost the latest version of his file. Each has lost the work which his current session at the console represents. He will have to backtrack and reconstruct everything he has done in this session (perhaps some previous sessions as well).

So extreme reliability becomes very important. In the second generation, batch mode of operation, a slight stutter by the computer has little effect; in the third generation, timesharing mode users consider such a stutter to be a major catastrophe. In this new generation, hardware and software together and cooperatively must be able to "fail soft" to minimize user frustration caused by even minor hardware failures.

In the field of machine organization, much work remains to be done. Hardware designers and software designers have long said that the two groups must design computers together. And they have, to a certain extent. But the computer world has still not come full cycle on the hardware/software design process. It is probably safe to say that there exists today no combined pair of packages, software and hardware, designed truly, intimately, and completely <u>for each other</u>. There have been a number of interesting changes

in machine organization which are the results of feedback from programmers. But we cannot honestly say that this design loop has been solidly closed.

The recently announced IBM 360/85 is an interesting milestone in this process. Characteristics of the Model 85 evolved by "running" present 360 programs (with present 360 software) on "paper" versions of the aborning Model 85. The final hardware organization, according to IBM, represents an economic tradeoff between the cost of indicated hardware features and how those features affect the operation of today's 360 applications programs running under today's 360 operating software.

Missing is the next step, or better, the next few iterative design cycles. What would happen if special compilers could be constructed to make efficient use of the new machine organization? How about determining which portions of the operating system dictate the most extensive hardware organizational changes. Can the operating system then be modified to eliminate the need for those hardware changes? Most applications programs these days are written in higher level languages and require a compilation step. Could we make substantial changes to machine organization which would make the compiler's job easier and faster? To make the compiled program smaller and faster?

Yes, some work has been done in this area, but much work remains.

* * * * * * *

So the software evolution continues, hidden beneath the more spectacular hardware revolution. The collection of many small, evolutionary changes in the software realm is certain to cause important changes in the entire computer field.